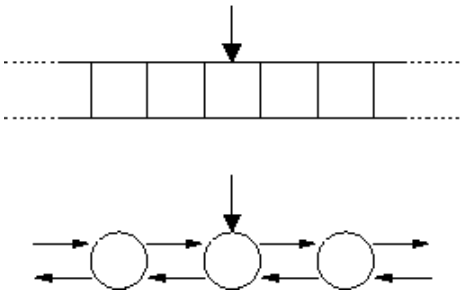
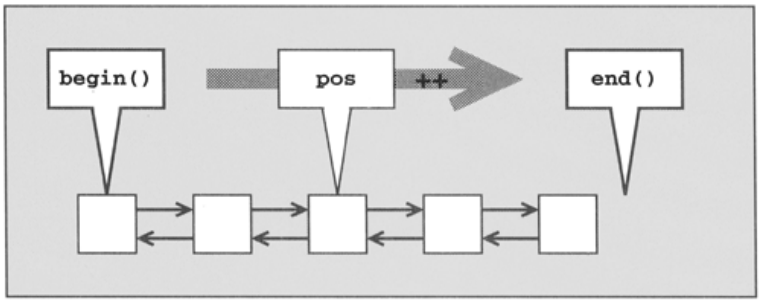
Iterator

- An iterator is a generalized pointer with a mechanism that lets us:
  - identify the position and access the elements in a container
  - navigate from one element to another
- Except for vector, modern C++ programs tend to use iterators rather than subscripts to access container elements.

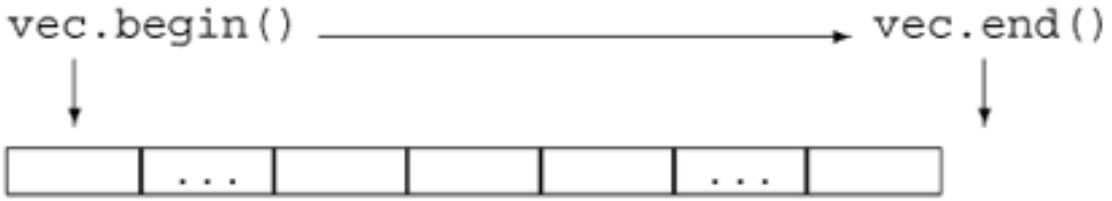




- Each container defines its own iterator type.

```
vector<int>::iterator iter;
vector<Sales_item>::iterator it;
set<double>::iterator it2;
```

- Each container defines a pair of functions begin and end that return iterators and cbegin and cend that return const_iterators.



vec.end(): an iterator positioned “one past the end”

- In general, we do not care the precise type of iterator

```
vector<int> vec; ...
auto b = vec.beign()
```

- Iterator is a pointer and it uses the dereference operator (the * operator) to access the element to which the iterator refers

```
*iter = 0;
```

- Iterators use the increment operator (++) to advance an iterator to the next element in the container.

```
++iter;
```

- Looping through a container using iterator.

- Each container type also defines a type named const_iterator for reading only.

See Note

Array

- An array consists of a type specifier (如int), an identifier (如myArray, yourArray), and a dimension.

- The type specifier indicates what **type** the elements stored in the array. The dimension specifies how many elements the array will contain.

```
int  intArray[10]; // an array of 10 ints

Sales_item  item[10]; // an array of 10 Sales_items
```

- Unlike vector, array has fixed size for better run-time performance (but at the cost of lost flexibility)

- The dimension must be a constant expression (see note).

Initializing Array Elements

```
int intArray[3] = {0, 1, 2}; // element initialization

int intArray[] = {0, 1, 2}; // element initialization

char ca1[] = {'C', '+', '+'}; // dim = 3
char ca2[] = {'C', '+', '+', '\0'}; // dim = 4
char ca3[] = "C++"; // dim = 4
```

char array is special

- If we do not supply explicit initialization, elements in an array are default initialized.

```
int intArray[3];
string sArray[3];
```

See Note

Operations on Arrays

```
const size_t array_size = 10;
int ia[array_size];
```

- (Q) How to assign value of each element equal to its index?

```
for (size_t i=0; i != array_size ; ++i)
    ia[i] = i;

for (auto i : ia)
    ia[i] = i;
```

- (Q) How to copy one array into the other?

```
int ia2[array_size];
```

Can we do ia2 = ia; why or why not?

Pointers and Arrays

- When we use the name of an array in an expression, that name is automatically converted into a pointer to the first element of the array:

```
int ia[] = {0,2,4,6,8};
int *ip = ia; // ip points to ia[0]
```

- We can use pointer arithmetic to compute a pointer to an element by adding (or subtracting) an integral value to (or from) a pointer to another element in the array:

```
int *ip2 = ip+4; // ip2 points to ia[4]
```

See Note